Um Arcabouço para Desenvolvimento de Algoritmos da Análise Formal de Conceitos

Sérgio M. Dias[1] e Newton J. Vieira[1]

Resumo: Este trabalho apresenta um arcabouço para o desenvolvimento de algoritmos da análise formal de conceitos (AFC). Seu principal componente é um gerador de contextos formais sintéticos cujos parâmetros podem ser ajustados de forma a permitir testar os algoritmos em situações de interesse. Presentes também no arcabouço estão alguns algoritmos básicos da AFC e alguns contextos formais obtidos a partir de bases de dados reais, desde um relativamente pequeno a um muito grande, este último normalmente impraticável em sua forma original. Para exemplificar a avaliação do desempenho de algoritmos via o arcabouço, seus algoritmos básicos são avaliados por meio de dados sintéticos e reais. Originalmente desenvolvido para servir como instrumento de auxílio ao aprendizado dos principais algoritmos da AFC, nada impede que o mesmo possa ser usado para avaliação de novos algoritmos. Para isto é de especial importância o fato de que os parâmetros na geração de dados são ajustáveis de forma a cobrir situações de interesse.

Abstract: This paper presents a framework for the development of formal concept analysis (FCA) algorithms. Its main component is a synthetic generator which formal parameters can be adjusted to allow testing the algorithms in situations of interest. It is also presented in the framework some basic FCA algorithms and some formal contexts obtained from real data sets, from a relatively small to a very large, the latter usually impractical at original form. To illustrate the algorithms performance evaluation via the framework, some basic algorithms are evaluated using synthetic and real data sets. Originally developed to serve as a tool to support the learning of the main FCA algorithms, there is nothing to prevent the framework from being used in the evaluation of new algorithms. For this, it's of particular importance the fact that the data generation parameters are adjustable to cover situations of interest.

1 Introdução

A análise formal de conceitos (AFC) é uma técnica da matemática aplicada baseada na matematização da noção de conceito e na estruturação dos conceitos em uma hierarquia conceitual[15]. Sua formalização teve início em 1982 com o trabalho apresentado por

[1]Departamento de Ciência da Computação - Universidade Federal de Minas Gerais (UFMG)
Av. Antônio Carlos, 6627 - ICEx - 4010 - Pampulha CEP: 31.270-010 - Belo Horizonte, MG, Brasil
{sergiomariano@gmail.com} e {nvieira@dcc.ufmg.br}

Wille[31], que propôs considerar cada elemento de um reticulado como um conceito formal e o reticulado como representando uma hierarquia entre os conceitos.

Atualmente existe um crescente interesse na aplicação da AFC em várias áreas, como recuperação de informação [6], mineração de dados [21], engenharia de software [5], redes neurais [34], redes sociais [20], ontologias [29], etc.

O grande potencial da AFC é devido, principalmente, [U+FFFD]rganização do conhecimento em uma hierarquia explícita. De fato, as principais aplicações fazem uso dessa hierarquia, que é constituída por um reticulado completo e representada por meio de diagrama de linhas, diagrama de linhas aninhado, árvores, etc.

O problema de gerar todos os conceitos formais e classificá-los hierarquicamente apresenta um comportamento exponencial no pior caso[22]. Apesar desse comportamento ser raramente encontrado em casos práticos [18], ainda assim o custo computacional pode ser proibitivo. Esse alto custo computacional justifica o estudo de algoritmos e abordagens alternativas para geração dos conceitos formais e construção do reticulado conceitual. Na literatura é encontrada uma grande quantidade de algoritmos com diferentes características[14, 16, 6, 24, 8, 17, 26, 4, 3, 28]. No intuito de facilitar a escolha dos algoritmos mais adequados para determinada aplicação, existem vários estudos comparativos[19, 17, 23, 13].

Kuznetsov e Obiedkov[23] fazem algumas críticas aos primeiros trabalhos de comparação de algoritmos. Segundo eles, o trabalho de Guénoche[19] não apresenta os algoritmos de uma forma correta nem supre todas as informações sobre os dados utilizados para testes. Já em relação ao trabalho de Godin et al.[17], eles observam que, como apenas algoritmos para construir o reticulado conceitual foram considerados, foram feitas alterações para complementar aqueles que originalmente geravam apenas os conceitos; no entanto, ressaltam Kuznetsov e Obiedkov, essas alterações podem aumentar o custo dos algoritmos drasticamente.

Quando se compara algoritmos para geração de conceitos formais ou reticulados, a utilização de dados sintéticos aleatórios ou de instâncias muito pequenas, como acontece em alguns trabalhos, pode não ser adequada, pois pode não ter correspondência nas situações reais em que os algoritmos serão aplicados. Segundo Fu e Nguifo[13], quando os algoritmos são aplicados em mineração de dados a comparação utilizando bases de dados reais pode ajudar a entender melhor o comportamento dos algoritmos em situações de interesse. Entretanto, nem sempre se tem acesso a bases reais (com os pré-processamentos requeridos) que espelhem todas as situações a serem encontradas. Assim, fica evidente a conveniência, não apenas de se ter um conjunto de bases de testes pré-estabelecidas, mas também de mecanismos para geração de bases com características semelhantes [U+FFFD]uelas das aplicações intencionadas. Um agravante citado por Kuznetsov e Obiedkov[23] é a inexistência de um processo de comparação entre os algoritmos bem definido e aceito pela comunidade.

Um outro problema é a inexistência de um ambiente computacional propício para o desenvolvimento e testes de algoritmos relacionados [U+FFFD]FC, em que seja possível o estudo e análise dos algoritmos em geral ou em situações que requeiram a escolha de algoritmos para aplicações específicas. Existem atualmente diversos aplicativos, demos e softwares para a AFC, dentre os quais destacam-se o Tockit project, ConExp, Galicia, Lattice Miner e OpenFCA[27].

O Tockit project é base de vários projetos como, por exemplo, o software ToscanaJ; ele é composto de um conjunto de componentes que podem ser utilizados para construção de novas ferramentas e algoritmos de forma prática. Já os softwares ConExp, Galicia e Lattice Miner são ferramentas que proporcionam a criação e manipulação de contextos formais, conceitos formais, reticulados conceituais e extração de regras. Em particular, o software Lattice Miner é voltado para mineração de dados e capaz de extrair regras de associação. Por último, OpenFCA é uma coleção de algoritmos da AFC implementados em linha de comando e que apresentam alto desempenho.

A principal limitação das ferramentas existentes é a dificuldade de implementação e testes de novos algoritmos; em outras palavras, é a ausência de um ambiente computacional que proporcione o estudo de algoritmos e metodologias de forma fácil e prática.

Dadas as considerações acima, este trabalho (a) apresenta um arcabouço didático para o desenvolvimento de algoritmos da AFC chamado *EF-Concept Analysis* (*Educational Framework for Concept Analysis*), o qual pode ser utilizado para estudo ou análise dos algoritmos de forma rápida e prática; (b) apresenta alguns algoritmos básicos da AFC para geração de conceitos e de reticulados conceituais, presentes no arcabouço, juntamente com a sua análise de complexidade; (c) propõe um conjunto de contextos formais obtidos a partir de bases de dados reais e um gerador de contextos formais sintéticos para testes de algoritmos; (d) avalia os algoritmos básicos por meio de *benchmarks* reais e sintéticos.

O restante deste artigo está organizado da seguinte forma. A seguir, na Seção 2, é apresentada uma pequena revisão da AFC no intuito de introduzir a terminologia e os aspectos notacionais. Os algoritmos básicos da AFC e respectivas análises de complexidade são exibidos na Seção 3. Na Seção 4 é esboçada a arquitetura do arcabouço EF-Concept Analysis para, em seguida, na Seção 5, ser abordada a metodologia de testes baseada em dados reais e sintéticos propiciada pelo arcabouço. Como estudo de caso, os algoritmos descritos na Seção 3 são testados segundo tal metodologia na Seção 6. Finalmente, seguem as conclusões na Seção 7.

2 Análise Formal de Conceitos - Uma pequena revisão

Esta pequena revisão, que também aparece em outro artigo dos mesmos autores[11], tem como propósito principal estabelecer a terminologia a ser empregada daqui para frente; ela é baseada essencialmente nas noções e terminologia expostas em [15].

A AFC é uma área da matemática que nasceu no início dos anos 80 [15, 6]. Sua característica principal é a representação do conhecimento por meio de diagramas específicos denominados diagramas de linha (ou diagramas de Hasse), que são o que os matemáticos chamam de diagramas de reticulados (conceituais). Em AFC, *contextos formais* são os elementos primordiais. Um contexto formal é uma tripla (G, M, I), em que G é um conjunto cujos elementos são denominados *objetos*, M é um conjunto cujos membros são chamados *atributos* e $I \subseteq G \times M$ é uma relação denominada *relação de incidência*. Se $(g, m) \in I$, diz-se que "*o objeto g tem o atributo m*". Um contexto formal é normalmente representado por uma tabela em que os objetos aparecem nos cabeçalhos das linhas, os atributos nos cabeçalhos das colunas e há uma marca na linha g e coluna m se e somente se $(g, m) \in I$.

Dado um conjunto de objetos $A \subseteq G$ de um contexto formal (G, M, I), pode-se perguntar que atributos em M são comuns a todos os objetos de A. Similarmente, pode-se perguntar, para um conjunto $B \subseteq M$, que objetos têm todos os atributos de B. Tais questões são respondidas pelos *operadores de derivação*, assim definidos: $A' = \{m \in M | \forall g \in A (g, m) \in I\}$ e $B' = \{g \in G | \forall m \in B (g, m) \in I\}$.

Um *conceito formal* é um par $(A, B) \in G \times M$ tal que $A' = B$ e $B' = A$, onde A é denominado a *extensão* e B *intenção* do conceito. O conjunto dos conceitos formais é ordenado pela ordem parcial $\leq$ tal que para quaisquer dois conceitos formais (A_1, B_1) e (A_2, B_2), $(A_1, B_1) \leq (A_2, B_2)$ se e somente se $A_1 \subseteq A_2$ (e, neste caso, $B_2 \subseteq B_1$). O conjunto de conceitos ordenado por $\leq$ constitui um reticulado completo[9] chamado *reticulado conceitual*. O reticulado conceitual obtido a partir de um contexto formal (G, M, I) é denotado por $\mathcal{B}(G, M, I)$.

O teorema básico sobre reticulados conceituais diz que um reticulado conceitual $\mathcal{B}(G, M, I)$ é um reticulado completo no qual para qualquer conjunto $C \subseteq \mathcal{B}(G, M, I)$ o *supremum* e o *infimum* são dados por $\bigwedge C = (\bigcap X, (\bigcup Y)'')$ e $\bigvee C = ((\bigcup X)'', \bigcap Y)$, sendo $X = \{A \,|\, (A, B) \in C\}$ e $Y = \{B \,|\, (A, B) \in C\}$[15].

3 Algoritmos Básicos da AFC

Existem duas tarefas básicas na AFC: geração dos conceitos formais a partir de um contexto formal e construção do reticulado conceitual, sendo que esta última pode ser feita diretamente a partir do contexto formal ou então a partir de conceitos formais previamente

obtidos.

Nas próximas seções serão apresentados algoritmos para gerar os conceitos formais a partir de um contexto formal (Seção 3.1), algoritmos para construir o reticulado conceitual a partir dos conceitos formais (Seção 3.2) e algoritmos para gerar os conceitos e construir o reticulado ao mesmo tempo (Seção 3.3), juntamente com suas complexidades.

3.1 Gerando Conceitos Formais

Antes da apresentação dos algoritmos básicos para gerar os conceitos formais, será exibido um algoritmo simples, porém ineficiente, no intuito de introduzir a necessidade da busca por melhores algoritmos.

Dado um contexto formal (G, M, I), sabe-se que um subconjunto A de G é a extensão de um conceito se, e somente se, $A'' = A$, e que um subconjunto B de M é a intenção de um conceito se, somente, se $B'' = B$. Como um conceito formal (A, B) é tal que $A' = B$ e $B' = A$, um algoritmo simples é o que gera (A'', A') para todo $A \subseteq G$ (ou (B', B'') para todo $B \subseteq M$). Para isto, os subconjuntos podem ser gerados em uma ordem pré-definida. Por exemplo, considere o conjunto de atributos M com n elementos e a existência de uma ordem total imposta aos elementos de M de tal forma que $M = \{m_1 < m_2 < ... < m_n\}$. Note que a ordem específica utilizada não é importante. Partindo-se da mesma, define-se uma ordem (lexicográfica) para os subconjuntos de M. Sejam $B_1, B_2 \subseteq M$. Então: $B_1 < B_2$ *se, somente, se* $\exists i\,(m_i \in B_2 - B_1)$ *e* $\forall j > i\,(m_i \in B_1 \Leftrightarrow m_j \in B_2)$.

O algoritmo *Next Set* (Algoritmo 1) recebe como entrada um conjunto $B \subseteq M$ e retorna o próximo conjunto segundo a ordem lexicográfica como definida acima. No caso em que $B = M$, retorna $\emptyset$.

Algoritmo 1 Algoritmo Next Set.

Input: Um conjunto B $\subseteq$ M.
Output: O conjunto seguinte a B pela ordem lexicográfica.

```
1: i = 0
2: while i < |M| do
3:    i = i+1
4:    if m_i ∉ B then
5:       return B ∪{m_i}
6:    end if
7:    B = B −{m_i}
8: end while
9: return ∅
```

O conjunto dos conceitos formais pode ser gerado pelo algoritmo *List of Concepts* (Algoritmo 2), que utiliza *Next Set* (na linha 4) para gerar cada subconjunto para, em seguida, gerar um conceito formal e adicioná-lo ao conjunto C de conceitos formais (na linha 5). O procedimento utilizado é ineficiente, já que o número de subconjuntos de um conjunto é exponencial.

Algoritmo 2 Algoritmo List of Concepts.

Input: Um contexto formal (G,M,I).
Output: O conjunto C de conceitos formais de $\mathcal{B}$(G,M,I).

```
1: B = G'
2: C = {(B',B)}
3: while B ≠ M do
4:     B = Next Set(B)
5:     C = C ∪ {(B',B'')}
6: end while
7: return C
```

Uma segunda proposta, mais eficiente, para gerar todos os conceitos formais é baseada na observação de que cada extensão de conceito é a interseção de extensões de atributos[15]. Assim, pode-se gerar todas as interseções de extensões de atributos e formar em seguida os conceitos formais a partir das extensões assim criadas. Tal procedimento é realizado pelo algoritmo *Intersection Sets* (Algoritmo 3). As extensões dos conceitos são obtidas nas linhas 2 a 5, enquanto que os conceitos são completados nas linhas 7 a 9. Uma estrutura de dados boa para implementação do conjunto de extensões, conjAs, manipulado nas linhas 3 e 4 do algoritmo, seria uma árvore *trie*, ou árvore de prefixos.

Algoritmo 3 Algoritmo Intersection Sets

Input: Um contexto formal (G,M,I).
Output: O conjunto C de conceitos formais de $\mathcal{B}$(G,M,I).

```
1: conjAs = {G}
2: for i = 1 to |M| do
3:     inters = {A ∩ {m_i}' | A ∈ conjAs}
4:     conjAs = conjAs ∪ inters
5: end for
6: C = ∅
7: for all A ∈ conjAs do
8:     C = C ∪ {(A,A')}
9: end for
10: return C
```

Para gerar todos os conjuntos nas linhas 2 a 5, o algoritmo *Intersection Sets* tem um custo de $O(|M||C|)$, excluindo-se a aplicação do operador de derivação na linha 3. Com esta, o custo total é de $O(|M||G||C|)$.

Historicamente, a primeira proposta de algoritmo para gerar conceitos formais data de 1962[2] com o trabalho de Malgrange[24], segundo Guénoche [19]. Entretanto, o primeiro algoritmo prático citado na literatura é o de Chein[8], que é uma melhoria do trabalho apresentado por Malgrange[24]. Ele gera os conceitos formais em um processo *bottom-up*, analisando o contexto formal (matriz) em camadas, em que a camada L_n representa todos os conceitos formais cuja intenção possui cardinalidade n. O algoritmo gera a extensão de cada conceito como a união de duas extensões de conceitos já existentes. A complexidade do algoritmo, segundo Kuznetsov e Obiedkov[23], é $O(|G|^3|M||L|)$, em que L é o número de camadas.[3]

3.1.1 Algoritmo Next Closure É considerado o mais importante para gerar conceitos formais. Como pode ser observado nos algoritmos anteriores, o mesmo conceito (na verdade, sua extensão ou intenção) pode ser gerado várias vezes. O algoritmo *Next Closure* (Algoritmo 4) busca justamente evitar isso.

Algoritmo 4 Algoritmo Next Closure

Input: Um contexto formal (G,M,I) e B $\subseteq$ M.
Output: O próximo conjunto fechado maior que B.

```
1:  i = 0
2:  while i<|M| do
3:     i = i+1
4:     if  m_i ∉ B then
5:        B = B ∪{m_i}
6:        if  ∄ j>i m_j ∈ B'' - B then
7:           return  B''
8:        end if
9:     end if
10:    B = B - {m_i}
11: end while
12: return  B
```

O algoritmo *Next Closure* foi proposto em 1984 por Ganter[14] com o objetivo de

[2]O leitor mais atento irá constatar que alguns algoritmos são datados anteriormente ao surgimento da AFC. Esses algoritmos foram posteriormente adaptados ou aplicados sob essa nova perspectiva.
[3]O objetivo original do algoritmo é gerar submatrizes máximas de uma matriz global, o que é equivalente a gerar conceitos formais.

encontrar conjuntos fechados, que é equivalente a obter conceitos formais. Ele usa uma ordem lexicográfica para os subconjuntos de atributos, como definido no início da Seção 3.1.

O algoritmo *Next Closure* usa o fato de que para qualquer conjunto B, se $B'' - B$ contém apenas elementos menores que os contidos no conjunto B, então todos os subconjuntos X entre B e B'' (segundo a ordem lexicográfica) são tais que $X'' = B''$ ou X'' é gerado posteriormente a partir um subconjunto após B''[6]. Desta forma é possível não examinar todos os subconjuntos do conjunto de atributos M. Como cada conjunto fechado é gerado uma única vez, não há necessidade de uma estrutura de busca auxiliar.

O algoritmo *All Concepts* (Algoritmo 5) utiliza o *Next Closure* para gerar os conceitos formais. Sua complexidade é O($|C||G||M|$), em que O(|G||M|) reflete o custo na computação dos operadores de derivação ($'$) e ($''$) e $|C|$ é o número de conceitos formais.

Algoritmo 5 Algoritmo All Concepts

Input: Um contexto formal (G,M,I).
Output: O conjunto C de conceitos formais de $\mathcal{B}$(G,M,I).

```
1: C = {(G,G')}
2: B = G'
3: while B ≠ M do
4:     B = Next Closure((G,M,I), B)
5:     C = C ∪ {(B',B)}
6: end while
7: return C
```

Apesar de ser um dos algoritmos mais indicados quando se quer gerar apenas os conceitos formais, nem sempre o *Next Closure* é o mais indicado quando é necessário gerar o reticulado conceitual, pois há o custo adicional para a ordenação dos conceitos, como visto a seguir.

3.2 Obtendo o Reticulado Conceitual a partir dos Conceitos Formais

Apesar de existirem aplicações para algoritmos que gerem apenas os conceitos formais, em muitos casos é necessário gerar também a relação de cobertura de modo a completar o reticulado conceitual. Em Godin et al.[17], são propostos alguns algoritmos que, em essência, percorrem a lista de conceitos com duas repetições aninhadas e comparam os conceitos dois a dois. Mais recentemente, Martin e Eklund[25] apresentam um algoritmo que possui a vantagem de não necessitar do contexto formal, já que não precisa aplicar os operadores de derivação.

3.2.1 Algoritmo Concepts Cover Proposto por Nourine e Raynaud[26], é um dos mais utilizados (Algoritmo 6). Em síntese, os descendentes de cada conceito são identificados através do cálculo de interseções (linha 7) em um processo semelhante ao do *Intersection sets* (Algoritmo 3). Para identificar, dentre os conceitos descendentes, os filhos do conceito atual, é utilizado um contador para cada conceito. O contador de um conceito descendente (A_1, B_1) é incrementado sempre que é detectado como candidato a filho do conceito (A, B) em questão; ele será considerado filho somente se seu contador for igual a $|B_1|-|B|$. A complexidade do algoritmo *Concepts Cover*, segundo Carpineto e Romano[6], é de O($|C||M|(|G|+|M|)$).

Algoritmo 6 Algoritmo Concepts Cover

Input: O conjunto C de conceitos formais e um contexto formal (G,M,I).
Output: Um reticulado conceitual L = $\mathcal{B}$(G,M,I).

```
1:  inicialize L com os conceitos em C.
2:  for all (A,B) ∈ C do
3:     for all c ∈ C do
4:        cont[c] = 0
5:     end for
6:     for all m ∈ M - B do
7:        A1 = A ∩ {m}'
8:        encontre (A1,B1) em C
9:        cont[(A1,B1)] = cont[(A1,B1)] + 1
10:       if ( |B1| - |B| ) = cont[(A1,B1)] then
11:          indique que (A1,B1) ≺ (A,B) em L
12:       end if
13:    end for
14: end for
15: return L
```

3.3 Obtendo o Reticulado Conceitual a partir do Contexto Formal

Na maioria das aplicações é necessário gerar o reticulado conceitual. O primeiro algoritmo capaz de gerar o reticulado conceitual completo em um único processo é o de *Bordat* [4]. Ele utiliza uma estratégia *top-down*, que inicia a geração dos conceitos a partir do supremo. Seguindo a ordem cronológica, tem-se depois os algoritmos de Godin et al.[16] e de Carpineto e Romano [7], que são algoritmos incrementais. Existem outros algoritmos, como *Dowling, Close by One, Lindidg, Nourine*[23] e extensões do algoritmo *Bordat*[3], mas a grande maioria são versões dos algoritmos anteriores.

3.3.1 Algoritmo Next Neighbours Um dos melhores e mais simples algoritmos conhecidos para construção do diagrama de linhas é o algoritmo *Next Neighbours*[6]. Ele pode ser considerado uma evolução do algoritmo *Bordat*. Iniciando do conceito supremo, (G, G'), o algoritmo constrói um nível por vez, onde o próximo nível é constituído dos filhos dos conceitos do nível atual, em um processo *top-down* (Algoritmo 7).

Algoritmo 7 Algoritmo Next Neighbours

Input: Um contexto formal (G,M,I).
Output: Um reticulado conceitual L = $\mathcal{B}$(G,M,I).

```
 1: C = {(G,G')}
 2: L = ∅
 3: nivAtual = {(G,G')} {nível inicial}
 4: while nivAtual ≠ ∅ do
 5:    nivProx = ∅
 6:    for all (A,B) ∈ nivAtual do
 7:       vizinhosInf = Find Lower Neigbours((A,B))
 8:       for all (A1,B1) ∈ vizinhosInf do
 9:          if (A1,B1) ∉ C then
10:             nivProx = nivProx ∪ {(A1,B1)}
11:          end if
12:          indique que (A1,B1) ≺ (A,B) em L
13:       end for
14:    end for
15:    C = C ∪ nivProx
16:    nivAtual = nivProx
17: end while
18: return L
```

Para calcular os filhos de um conceito formal é utilizado o algoritmo *Find Lower Neighbours* (Algoritmo 8). Ele é baseado no fato de que os vizinhos inferiores de um conceito (A, B) são formados pela adição de novos atributos. Segundo Carpineto e Romano[6], o algoritmo tem um custo de $O(|C||G||M|^2)$.

Em muitas aplicações é necessária a atualização do contexto formal e, consequentemente, do reticulado conceitual, em consequência da inserção ou exclusão de objetos ou atributos (a inserção de objetos é a mais comum).

O primeiro algoritmo incremental foi proposto por Godin em 1991[4]. Posteriormente, apareceram os algoritmos de Carpineto e Romano[7] e Qiao et al.[28], que são adaptações

[4]Na literatura o algoritmo *Godin* também é conhecido por *algoritmo de atualização pela cardinalidade*, *update by cardinality algorithm* em inglês.

Algoritmo 8 Algoritmo Find Lower Neighbours

Input: Um conceito formal (A,B).
Output: O conjunto de conceitos formais filhos de (A,B).

```
1: filhos = ∅
2: for all m ∈ M − B do
3:     A1 = (B ∪ {m})′
4:     B1 = A1′
5:     filhos = filhos ∪ {(A1,B1)}
6: end for
7: return filhos
```

do algoritmo de Nourine capazes de atualizar o reticulado. Já em Valtchev e Missaoui [30] é apresentada uma proposta que permite a inserção de um conjunto de objetos ao mesmo tempo.

3.3.2 Algoritmo Godin Godin et al.[16] apresentam um algoritmo (Algoritmo 9) que atualiza o reticulado conceitual após a inserção de um novo objeto. Para um objeto h, os novos conceitos introduzidos são da forma $(A \cup \{h\}, B \cap \{h\}')$ para algum conceito (A, B) presente no reticulado (linha 18). Para evitar a necessidade de analisar todos os conceitos formais eles são agrupados pela cardinalidade da intenção (linha 2).

Além de atualizar os conceitos formais, é necessário atualizar a relação de cobertura. Quando um novo conceito formal é criado ele deve ser ligado a conceitos já existentes ou não existentes. Já quando um conceito é apenas atualizado seu pai não é atualizado, mas seus filhos podem sofrer alterações.

Godin et al.[16] afirmam que, apesar da complexidade ser exponencial no número de conceitos, na prática esse comportamento não é observado. Segundo os autores, a complexidade do algoritmo é de $O(|G|2^K)$, em que K é o número de atributos do objeto que possui a maior quantidade de atributos.

Note que o algoritmo descrito trata apenas da inserção de objetos. Evidentemente, há o dual para inserção de atributos, cujo uso é menos frequente. Carpineto e Romano[6] descrevem algoritmos para cada atualização possível (inserção ou deleção de atributos e inserção ou deleção de objetos).

4 O Arcabouço EF-Concept Analysis

Como enfatizado na Introdução, a comunidade que trabalha com AFC disponibiliza vários softwares e ferramentas[27], como, por exemplo, os softwares ToscannaJ e Conexp.

Algoritmo 9 Algoritmo Godin

Input: Um reticulado conceitual L = $\mathcal{B}$(G,M,I), um novo objeto h e os atributos h$'$.
Output: O reticulado conceitual L atualizado.

```
1:  for j = 0 to |M| do
2:     grupoAnt[j] = {(A,B) ∈ L | |B| = j}
3:     grupoNovo[j] = ∅
4:  end for
5:  for j = 0 to |M| do
6:     for all (A,B) ∈ grupoAnt[j] do
7:        if B ⊆ {h}' then
8:           substitua (A,B) em L por (A ∪ {h}, B)
9:           grupoNovo[j] = grupoNovo[j] ∪ {(A∪{h},B)}
10:          if B = {h}' then
11:             return L
12:          end if
13:       else
14:          inters = B ∩ {h}'
15:          if {(A1,B1) ∈ grupoNovo[|inters|] | B1 = inters} = ∅ then
16:             concNovo = (A ∪ {h}, inters)
17:             grupoNovo[|inters|] = grupoNovo[|inters|] ∪ {concNovo}
18:             indique que (A,B) ≺ concNovo em L
19:             for k = 0 to (|inters| - 1) do
20:                for all (A1,B1) ∈ grupoNovo[k] do
21:                   if B1 ⊂ inters e ∄ (A2,B2) filho de (A1,B1) tal que B2 ⊂ inters then
22:                      indique que concNovo ≺ (A1,B1) em L
23:                      if (A1,B1) é pai de (A,B) then
24:                         remova (A,B) ≺ (A1,B1) em L
25:                      end if
26:                   end if
27:                end for
28:             end for
29:             if inters = {h}' then
30:                return L
31:             end if
32:          end if
33:       end if
34:    end for
35: end for
36: return L
```

Contudo, não existem softwares que proporcionem a estudantes e profissionais um ambiente de desenvolvimento em que se possa implementar algoritmos e analisar o desempenho dos mesmos em diferentes situações de interesse.

Nesta seção, é proposto um arcabouço didático para o desenvolvimento de algoritmos da AFC chamado *EF-Concept Analysis* (*Educational Framework for Concept Analysis*)[5]. Esse arcabouço foi desenvolvido em Java utilizando o paradigma de orientação por objeto. Além disso, todos os serviços foram orientados por interface, de forma que o arcabouço pudesse ser estendido facilmente. O arcabouço *EF-Concept Analysis* é composto de oito pacotes com as seguintes funcionalidades:

- *io*: responsável por entrada e saída dos dados. Possui uma interface para leitura do contexto formal no formato *burmeister*, que é amplamente utilizado pela comunidade. Para exibir o reticulado conceitual é utilizado um formato XML chamado GSH (*Galicia Sub-Hierarchy*) lido pelo software Galicia[6]. Observe que outros formatos podem sem ser facilmente utilizados, simplesmente implementando-se as interfaces definidas.

- *set*: possui uma interface definindo operações sobre conjuntos. Também possui classes implementando conjuntos simples e ordenados.

- *statistic*: possui um conjunto de classes e métodos para colher estatísticas do reticulado conceitual e para análise da relação entre os dados utilizados para construir o reticulado conceitual e um conjunto de regras extraídas do reticulado.

- *exception*: conjunto de exceções lançadas pelo arcabouço.

- *context*: define uma interface para um contexto formal e uma estrutura de dados baseada em cadeia de bits para representar o contexto. Também são definidos tipos abstratos de dados para objetos, atributos e conceitos formais.

- *utilities*: conjunto de classes úteis, dentre elas destacando-se uma classe para produção de contextos formais a partir de dados reais.

- *algorithms*: conjunto de algoritmos e exemplos de utilização do arcabouço. O arcabouço possui implementados os algoritmos *Next Closure*, *Concepts Cover*, *Next Neighbours*, *Frequent Next Neighbours*, *Godin* e *Find Implications*.

- *benchmark*: interfaces e classes para produção de contextos formais sintéticos.

[5]Disponível em: http://www.dcc.ufmg.br/~mariano
[6]Disponível em: http://www.iro.umontreal.ca/~galicia

5 Benchmark para Avaliação dos Algoritmos

Nesta seção é apresentado um conjunto de contextos formais sintéticos e reais, que buscam mostrar como o desempenho dos algoritmos básicos da AFC abordados na seção anterior pode ser analisado experimentalmente. Outros algoritmos eventualmente desenvolvidos e agregados ao arcabouço podem ser comparados com estes mediante os mesmos dados sintéticos e/ou reais.

5.1 Benchmark Sintético

O número de conceitos formais e a estrutura propiciada pela relação de cobertura obtidos a partir de um contexto formal são influenciados pelo número de objetos, número de atributos, densidade da relação de incidência e pela forma de distribuição da densidade. Portanto, um conjunto de contextos formais sintéticos para avaliação dos algoritmos deve considerar esses parâmetros. Na literatura existem poucos trabalhos sobre contextos formais sintéticos, mas, em geral, os trabalhos que propõem ou utilizam esse tipo de contexto procuram variar esses parâmetros, como os apresentados por Arévalo et al.[2] e Kuznetsov e Obiedkov[23], dentre outros.

Uma segunda opção, muito utilizada, é o software *IBM synthetic data generator*[7], o qual implementa a metodologia proposta por Agrawal et. al.[1]. Em síntese, Agrawal et. al.[1] propõem uma base de dados sintética voltada para mineração de dados, mais precisamente para extração de regras de associação, que espelha-se no comportamento de bases de dados transacionais. Uma crítica quanto [U+FFFD]tilização dessa base de dados sintética é o fato de seu comportamento ser característico de um tipo de aplicação. Ao mudar o foco da aplicação, por exemplo, para recuperação de informação, bioinformática, redes sociais, etc, a distribuição de incidências no contexto formal será diferente, o que provavelmente irá induzir um comportamento diferente. Desta forma, esse trabalho tem como objetivo a proposta de um gerador de *benchmark* sintético de uso geral, que possa ser utilizado para testes de algoritmos da AFC. A seguir, o uso de tal gerador será exemplificado para teste dos algoritmos vistos na seção anterior, para isto mostrando como produzir vários tipos de contextos formais sintéticos que explorem diferentes características.

Antes de apresentar as características dos contextos formais propostos para teste dos algoritmos, é conveniente ressaltar que os valores sugeridos abaixo para número de objetos, número de atributos e densidade foram baseados em experiência do primeiro autor no desenvolvimento de aplicações[34, 33, 10, 11] e do próprio arcabouço *EF-Concept Analysis*. Assim, eles podem ser considerados valores típicos em contextos formais para ampla gama de aplicações possíveis. Isso, no entanto, não significa que não possam surgir aplicações para

[7] Disponível em:http://www.cs.rpi.edu/~zaki/software/.

as quais outros valores sejam mais adequados para avaliação desses mesmos algoritmos ou de outros.

Mais importante do que os valores específicos sugeridos é o processo de testes proposto, o qual procura cobrir uma gama a mais ampla possível de situações observadas, sendo ele plenamente suportado pelo arcabouço. Em geral, os testes de desempenho dos algoritmos da AFC observados na literatura não apresentam essa preocupação de cobrir uma ampla gama de situações. Por exemplo, não é comum realizar testes utilizando uma quantidade maior de objetos do que atributos. Contudo, como ressaltado por Fu e Nguifo[12], frequentemente é observado um desempenho diferente para contextos formais com diferentes características. Segue a proposta de aspectos para os testes e valores de parâmetros respectivos. Considere contextos formais (G, M, I).

1. Explore as situações de pior caso para os algoritmos, quais sejam, aquelas em que os contextos formais têm apenas a diagonal principal não preenchida. Valores utilizados: $|M| = |G| = 2, 3, 4, \ldots, 20$.

2. Crie contextos formais com várias densidades de I, sendo seus elementos distribuídos aleatoriamente[8]. Para cada valor de densidade, utilize poucos objetos e muitos atributos e muitos objetos e poucos atributos. Foram utilizadas as densidades 5, 10, 15, ..., 95%, cada uma com $|G| = 20$ e $|M| = 50$, e com $|G| = 50$ e $|M| = 20$.

3. Faça como no item anterior, distribuindo as incidências aleatoriamente, mas com o mesmo número de atributos por objeto[9]. Foram utilizados os mesmos números que no item anterior.

4. Para uma quantidade de atributos e uma densidade de I fixas, varie o número de objetos, sendo os elementos de I distribuídos aleatoriamente. Valores utilizados: $|M| = 20$, $|I| = 50\%$, $|G| = 20, 40, 60, \ldots, 200$.

5. Para uma quantidade de objetos e uma densidade de I fixas, varie o número de atributos, sendo os elementos de I distribuídos aleatoriamente. Valores utilizados: $|G| = 20$, $|I| = 50\%$, $|M| = 5, 10, 15, \ldots, 100$.

O primeiro item procura levar o algoritmo ao extremo. Alguns autores, como Kuznetsov e Obiedkov[23], Godin et al.[17] e Carpineto e Romano[6] comentam, com razão, que esse comportamento é raramente observado em problemas práticos. Contudo, saber qual é o limite superior de um algoritmo é extremamente relevante por revelar, por exemplo, um limiar a partir do qual o uso apenas de memória primária não é mais possível.

[8] A densidade de I é $|I|/(|G| \times |M|)$.
[9] Neste caso, $|I|/(|G| \times |M|) = |\{(g, m) \mid (g, m) \in I\}|/|M|$ para cada $g \in G$.

O parâmetro que mais influencia no número de conceitos formais é a densidade de $|I|$ e como ela está distribuída; daí, a proposta de variá-la. Em relação [U+FFFD]istribuição da densidade de I, sugere-se a utilização de uma distribuição aleatória na qual cada objeto pode possuir qualquer número de atributos, e outra, na qual cada objeto possui o mesmo número de atributos. Como as quantidades de objetos e de atributos também influenciam no número de conceitos formais, propõe-se a combinação desses parâmetros na forma de muitos objetos e poucos atributos, e muitos atributos e poucos objetos.

Por último, propõe-se a variação do número de objetos e atributos para uma dada densidade. Note que a variação do número de objetos proposta é maior do que a de atributos, e que a variação do número de atributos tem uma granularidade menor. Essa diferença espelha-se em contextos reais, que frequentemente apresentam uma quantidade maior de objetos que atributos.

5.2 Benchmark Real

Uma questão desafiadora é: como escolher um conjunto de dados reais realmente significativos para avaliação dos algoritmos? Os trabalhos apresentados na literatura que têm como objetivo apresentar algum algoritmo e/ou realizar comparações entre algoritmos, são constantemente criticados quanto [U+FFFD] características das bases de dados reais utilizadas. Em geral observam-se críticas sobre a qualidade dos dados, dimensões, número de registros, não disponibilidade, falta de descrição dos contextos formais construídos, dentre outras.

Assim, como parte do *benchmark* proposto para avaliação dos algoritmos é apresentado e caracterizado, nessa subseção, um conjunto de contextos formais construídos a partir de dados reais. Foram selecionadas quatro bases de dados clássicas da UCI (*Machine Learning Repository*)[10], são elas: *Iris Data Set, Wine Data Set, Yeast Data Set* e *Water Treatment Plant Data Set* (elas serão referenciadas daqui para frente por Íris, Wine, Yeast e Water, respectivamente). As três primeiras estão entre as dez principais de um total de 177 bases de dados segundo índice de popularidade mantido pelo repositório desde 2007. As bases foram selecionadas seguindo o critério de popularidade, número de registros e número de atributos, sendo o de maior número de utilizações considerado o mais relevante. Na Tabela 1 é apresentado um resumo das características dos contextos formais obtidos. O processo de limpeza dos dados e construção dos contextos formais é descrito detalhadamente em [10]. Foram escolhidos contextos formais que redundassem desde poucos a muitos conceitos formais.

[10]Disponível em: http://www.ics.uci.edu/~mlearn.

Tabela 1. Resumo das características das bases de dados reais que compõem o *benchmark*.

	Contextos Formais			
	Íris	**Wine**	**Yeast**	**Water**
Número de objetos	18	132	110	316
Número de atributos	12	29	27	76
Número médio de atributos por objeto	4.8%	14%	9%	34.9%
Densidade	40.3%	48.2%	33.3%	52.1%
Número de Conceitos Formais	104	29032	1930	8940648

6 Estudo de caso

A seguir são apresentadas as avaliações dos algoritmos básicos apresentados na Seção 3, implementados na framework *EF-Concept Analysis*, através dos benchmarks sintético e real definidos na seção anterior.

6.1 Avaliações pelo Benchmark Sintético

Na Tabela 2 está mostrado o desempenho de cada algoritmo para o aspecto correspondente ao item 1 de avaliação, qual seja, o de contextos formais de pior caso (apenas a diagonal principal não preenchida). A notação "*" indica que a memória principal disponível não foi suficiente para gerar os conceitos formais. O tempo gasto está medido em horas (hh), minutos (mm), segundos (ss) e milissegundos (ms).

Na última coluna da Tabela 2 constata-se o comportamento exponencial com relação ao número de conceitos formais. Note que a partir de $|G| = |M| = 18$ apenas o algoritmo *Next Closure*, cujo desempenho está mostrado na coluna 1, foi capaz de gerar os conceitos formais. Na literatura, esse algoritmo é citado como um dos melhores, principalmente para contextos formais densos. No entanto, para $|G| = |M| = 20$ ele gastou quase duas horas para gerar os 2^{20} conceitos formais. A tabela indica que contextos formais com $|G| > 18$ ou $|M| > 18$ e alta densidade necessitam, para o arcabouço *EF-Concept Analysis*, do uso de memória secundária.

Como o algoritmo *Next Closure* fornece apenas os conceitos formais, o algoritmo *Concepts Cover*, cujo desempenho está mostrado na coluna 2, pode ser utilizado para ordenar os conceitos, formando assim o reticulado conceitual. Note, na coluna 3, que o custo total dos dois algoritmos foi muito alto, quando comparado com aquele gasto pelo algoritmo *Next Neighbours* (coluna 4), sendo a maior parte desse custo proveniente do algoritmo *Concepts Cover*.

Tabela 2. Desempenho dos algoritmos para contextos de pior caso.

	hh:mm:ss:ms					
Objetos e Atributos	**Next Closure**	**Concepts Cover**	**Next Closure + Concepts Cover**	**Next Neighbours**	**Godin**	**No. de Conceitos Formais**
2	00:00:00:001	00:00:00:004	00:00:00:005	00:00:00:002	00:00:00:002	4
3	00:00:00:001	00:00:00:001	00:00:00:002	00:00:00:000	00:00:00:000	8
4	00:00:00:001	00:00:00:002	00:00:00:003	00:00:00:001	00:00:00:001	16
5	00:00:00:002	00:00:00:003	00:00:00:005	00:00:00:004	00:00:00:002	32
6	00:00:00:002	00:00:00:003	00:00:00:005	00:00:00:005	00:00:00:004	64
7	00:00:00:003	00:00:00:006	00:00:00:009	00:00:00:008	00:00:00:005	128
8	00:00:00:006	00:00:00:021	00:00:00:027	00:00:00:011	00:00:00:010	256
9	00:00:00:007	00:00:00:083	00:00:00:090	00:00:00:022	00:00:00:034	512
10	00:00:00:018	00:00:00:336	00:00:00:354	00:00:00:053	00:00:00:122	1024
11	00:00:00:049	00:00:01:171	00:00:01:220	00:00:00:094	00:00:00:410	2048
12	00:00:00:140	00:00:05:450	00:00:05:590	00:00:00:213	00:00:01:677	4096
13	00:00:00:455	00:00:13:012	00:00:13:467	00:00:00:526	00:00:07:194	8192
14	00:00:01:748	00:00:47:180	00:00:48:928	00:00:01:472	00:00:28:068	16384
15	00:00:06:128	00:03:52:738	00:03:58:866	00:00:03:794	00:01:50:554	32768
16	00:00:25:451	00:17:54:059	00:18:19:510	00:00:11:019	00:08:13:023	65536
17	00:01:50:947	*	*	00:00:31:160	00:31:10:268	131072
18	00:06:37:575	*	*	*	*	262144
19	00:26:26:298	*	*	*	*	524288
20	01:46:26:825	*	*	*	*	1048576

O desempenho com relação ao item 2 de avaliação, contemplando contextos formais com as densidades 5, 10, 15, ..., 95% para I, está mostrado nas Figuras 1 e 2 para $|G| = 20$ e $|M| = 50$, e na Figura 3 para $|G| = 50$ e $|M| = 20$. A Figura 1 mostra o comportamento do número de conceitos formais e as Figuras 2 e 3 mostram o tempo gasto. Note que fixando a densidade e utilizando $|G| = 20$ e $|M| = 50$ ou $|G| = 50$ e $|M| = 20$ o mesmo número de conceitos formais será obtido. Pois, nesse caso, tem-se um contexto formal dual. Contudo, é relevante a análise do comportamento dos algoritmos, pois frequentemente é observado um comportamento diferente para um contexto formal e seu dual[13].

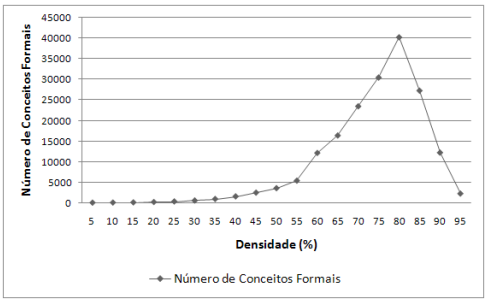

Figura 1. Número de conceitos para $|G|$ = 20, $|M|$ = 50 e uma distribuição aleatória.

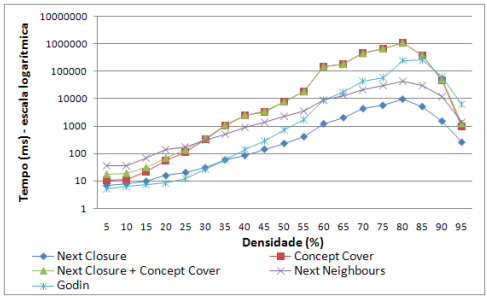

Figura 2. Tempo em milissegundos para $|G|$ = 20, $|M|$ = 50 e uma distribuição aleatória.

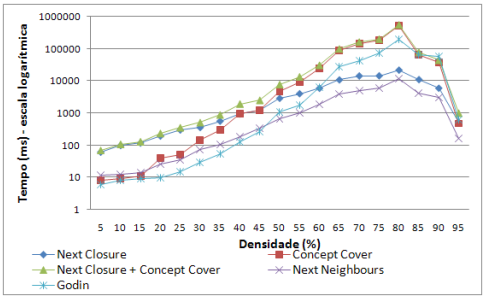

Figura 3. Tempo em milissegundos dos paradigmas para $|G|$ = 50, $|M|$ = 20 e uma distribuição aleatória.

Inicialmente, observe que as Figuras 2 e 3 revelam que o pior desempenho para gerar o reticulado é da dupla *Next Closure* e *Concepts Cover*. Note também que mais tempo é gasto no processo de ordenação dos conceitos formais pelo algoritmo *Concepts Cover* do que na geração de conceitos pelo *Next Closure*. Observe ainda que para densidades abaixo de

35% os gráficos mostram que o algoritmo *Godin* é mais indicado. À medida que a densidade aumenta, até aproximadamente 80%, o desempenho do algoritmo *Godin* torna-se pior, o que é compreensível, visto que a cada novo objeto inserido é necessário atualizar uma grande quantidade de conceitos formais.

Os gráficos 1, 2 e 3 mostram ainda que há um decréscimo do número de conceitos formais e do tempo para geração dos conceitos após uma densidade de aproximadamente 80%. À medida que a densidade aumenta, começam a surgir objetos que possuem todos os atributos e atributos que se aplicam a todos os objetos, o que tende a reduzir o número de conceitos formais.

A sequência de gráficos apresentada nas Figuras 4, 5 e 6 mostra o desempenho com relação ao item 3 de avaliação, contemplando contextos formais com as densidades 5, 10, 15, ..., 95% para I, com o mesmo número de atributos por objeto, considerando $|G| = 20$ e $|M| = 50$ (Figuras 4 e 5) e $|G| = 50$ e $|M| = 20$ (Figura 6). Note que não é observado um decréscimo no número de conceitos formais, ao contrário do que ocorre na Figura 1. Isso acontece porque no presente caso não são gerados objetos que possuam todos os atributos. Com relação ao desempenho, o comportamento é similar ao observado nas Figuras 2 e 3.

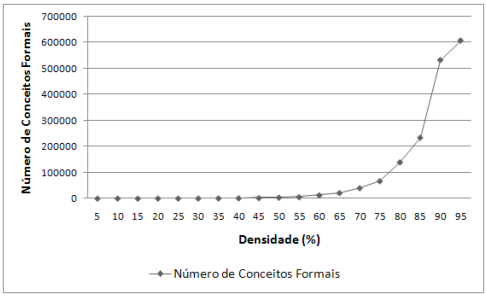

Figura 4. Número de conceitos para $|G|$ = 20, $|M|$ = 50 e uma densidade média.

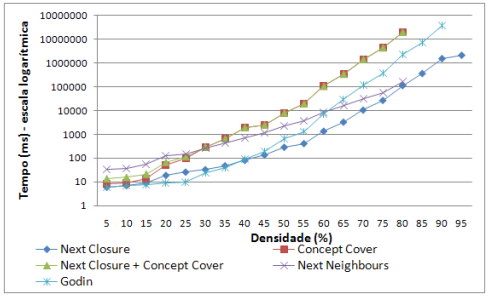

Figura 5. Tempo em milissegundos dos paradigmas para $|G|$ = 20, $|M|$ = 50 e uma densidade média.

Nos gráficos apresentados nas Figuras 7 e 8 é exibido o comportamento para o item 4 de avaliação, em que se considera $|M| = 20$, $|I| = 50\%$ e $|G| = 20, 40, 60, \ldots, 200$. Nota-se que o crescimento do número de conceitos é menor para uma variação do número de objetos do que para a variação da densidade mostrada nos gráficos das Figuras 1 e 4, o que confirma as observações encontradas na literatura[6]. Mais uma vez o desempenho da dupla *Next Closure* e *Concepts Cover* foi o pior.

Finalmente, a Figura 9 mostra o resultado para o item 5 de avaliação, que considera

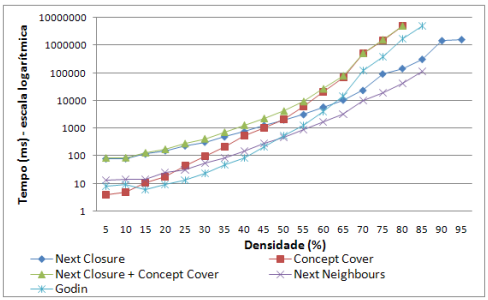

Figura 6. Tempo em milissegundos dos paradigmas para $|G| = 50$, $|M| = 20$ e uma densidade média.

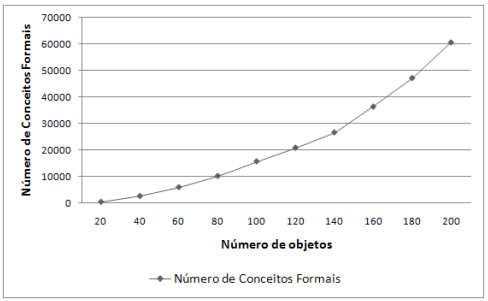

Figura 7. Número de conceitos para $|M| = 20$, $|I| = 50\%$ aleatória e uma variação do número de objetos.

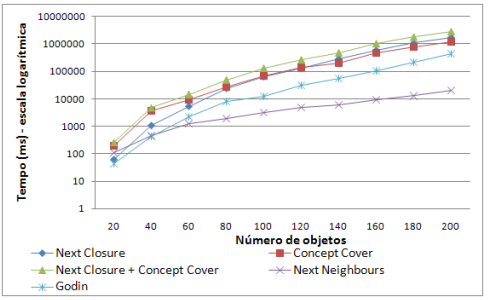

Figura 8. Tempo em milissegundos dos paradigmas para $|M| = 20$, $|I| = 50\%$ aleatória e uma variação do número de objetos.

$|G| = 20$, $|I| = 50\%$ e $|M| = 5, 10, 15, \ldots, 100$. No mundo real raramente é encontrada uma aplicação em que o número de objetos varia (situação em que o algoritmo *Godin* se aplicaria). Entretanto, o teste descrito é relevante por demonstrar o comportamento dos algoritmos nessa configuração. Caso o algoritmo em análise apresente melhores resultados pode-se, por exemplo, trabalhar com um contexto formal dual (invertendo objetos e atributos). O tempo gasto pelo par *Next Closure* e *Concepts Cover* novamente foi muito elevado. Nas condições em análise é mais adequada a utilização do algoritmo *Godin*. Note que o algoritmo de *Godin* foi melhor devido ao fato do número de objetos não ser elevado.

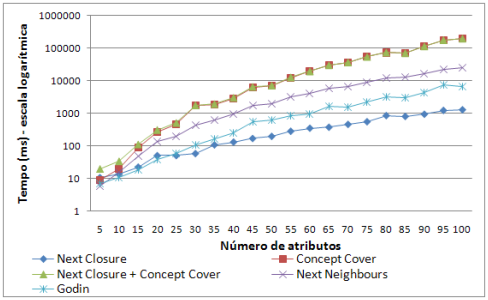

Figura 9. Tempo em milissegundos dos paradigmas para $|G| = 20$, $|I| = 50\%$ aleatória e uma variação do número de atributos.

6.2 Avaliações pelo Benchmark Real

Na Tabela 3 são apresentados os tempos de execução para as bases de dados reais do *benchmark* real. Considerando as características dos contextos formais, apresentadas na Tabela 1, seguem alguns comentários.

Como pode ser observado na última linha da Tabela 3, para o contexto formal Water, que possui quase 9 milhões de conceitos formais, nenhum algoritmo foi capaz de obter os conceitos formais após oito horas de execução[11]. Mesmo possuindo um contexto formal com apenas 316 objetos, 76 atributos e uma densidade de 52.1%, o número conceitos formais é elevado, o que inviabilizou os algoritmos. Nesse caso é indicado a aplicação de técnicas de redução da complexidade de reticulados conceituais[10, 11].

Tabela 3. Aplicações do *benchmark* real.

	hh:mm:ss:ms				
Contextos Formais	**Next Closure**	**Concepts Cover**	**Next Closure + Concepts Cover**	**Next Neighbours**	**Godin**
Íris	00:00:00:071	00:00:00:052	00:00:00:123	00:00:00:051	00:00:00:052
Wine	00:01:10:727	00:09:35:070	00:10:45:797	00:00:06:054	00:00:21:272
Yeast	00:00:03:159	00:00:00:716	00:00:03:875	00:00:00:258	00:00:00:422
Water	*	*	*	*	*

Para a base de dados Yeast, a Tabela 3 mostra que o maior tempo para a construção do reticulado conceitual foi novamente da dupla *Next Closure* e *Concepts Cover*, como esperado. Entretanto, ao contrário do comportamento observado para os dados sintéticos, o

[11] O número total de conceitos formais foi obtido utilizando o software Conexp[32].

maior custo da dupla foi do algoritmo *Next Closure*. O fato do contexto formal gerado para a base de dados Yeast apresentar uma baixa densidade e, consequentemente, poucos conceitos formais, torna mais custosas as operações de derivação realizadas pelo *Next Closure* do que o processo de ordenação dos conceitos formais realizado pelo *Concepts Cover*. Um meio de diminuir esse custo é manter o contexto formal ordenado pelo número de incidências em cada objeto, e uma cópia do contexto formal ordenado pelo número de incidências em cada atributo, semelhante [U+FFFD]roposta em [12].

Para o contexto formal representando a base de dados Wine, o melhor desempenho para obtenção do reticulado conceitual foi do algoritmo *Next Neighbours*. Note que o contexto formal para essa base de dados possui uma quantidade de objetos muito superior [U+FFFD]e atributos ($|G| = 132$ e $|M| = 29$).

Finalmente, para o contexto formal produzido para a base de dados Íris, todos os algoritmos apresentaram um comportamento similar para construção do reticulado conceitual, sendo o pior tempo do par *Next Closure* e *Concepts Cover* e o melhor do algoritmo *Next Neighbours*.

Conforme já comentado, um algoritmo projetado para ter alto desempenho para certo grupo de aplicações pode superar os algoritmos aqui analisados para alguns dos contextos formais apresentados nesta seção, sejam eles reais ou sintéticos. Entretanto, cabe ressaltar que o objetivo aqui é demonstrar o processo de análise. Ao *benchmark* proposto pode-se adicionar novos contextos formais, formando assim um *benchmark* mais amplo, capaz de melhor avaliar algoritmos para situações de interesse.

7 Conclusões

Neste trabalho foi apresentado um arcabouço para o desenvolvimento e testes de algoritmos da AFC. Além de ter alguns algoritmos básicos, ele tem como componentes benchmarks sintéticos e reais, sendo que benchmarks sintéticos podem ser gerados para o teste de desempenho de algoritmos novos ou existentes a partir da atribuição de valores a parâmetros. Esta última característica é de especial importância para permitir a cobertura das situações de interesse.

Foram apresentados e analisados algoritmos básicos da AFC para geração de conceitos formais e construção de reticulados conceituais: *Next Closure*, *Concepts Cover*, *Next Neighbours* e *Godin*. Em síntese, o algoritmo *Next Closure* é o algoritmo padrão para construção dos conceitos formais quando não há necessidade de ordená-los para completar o reticulado conceitual. O algoritmo *Concepts Cover* é o comumente indicado para a ordenação dos conceitos formais, quando se deseja construir o reticulado conceitual a partir de um conjunto prévio de conceitos formais. Para gerar o reticulado conceitual diretamente a partir do con-

texto formal, o algoritmo *Next Neighbours* é o indicado. Por último, o algoritmo *Godin* é um representante do paradigma incremental, quando se deseja inserir novos objetos a um reticulado já existente.

Foram também apresentados dois tipos de *benchmark* para avaliação de algoritmos da AFC. Um deles exemplifica a geração de bases de dados sintéticas, tendo sido estas inspiradas no que comumente é considerado importante na literatura. Desta forma, tal *benchmark* sintético procurou explorar os algoritmos para o pior caso e para variações da densidade, número de objetos e números de atributos. O outro *benchmark* foi obtido de bases reais muito utilizadas pela comunidade.

Na aplicação dos benchmarks aos algoritmos estudados, verificou-se que o algoritmo *Next Closure* gasta, para a geração dos conceitos formais, um tempo menor do que aquele gasto pelo algoritmo *Concepts Cover* para ordenação dos conceitos. E a combinação dos dois para construção de um reticulado conceitual é normalmente menos eficiente do que o uso do *Next Neighbours*, um algoritmo que obtém o reticulado diretamente a partir do contexto formal. Na situação de pior caso, o algoritmo *Godin* só é melhor do que o *Next Neighbours* quando há muito poucos objetos e atributos, mas neste caso os três tipos de algoritmos são praticamente equivalentes em desempenho. Embora tais resultados sejam fundamentados na implementação do arcabouço *EF-Concept Analysis*, os resultados são os esperados.

Um ponto que pode ser questionado é a dimensão (número de registros e atributos) das bases de dados sintéticas e reais utilizadas para análise dos algoritmos feita neste trabalho (no mundo real uma alta dimensionalidade é frequentemente observada). Entretanto, o objetivo principal aqui é apresentar, junto com alguns algoritmos básicos da AFC, uma forma bem definida de análise de algoritmos existentes e/ou novos no contexto de um arcabouço em que tal análise seja factível de ser concebida e levada a efeito.

Como trabalhos futuros pretende-se: 1º) incorporar ao arcabouço funcionalidades gráficas para manipulação de contextos formais, conceitos formais, reticulados conceituais, conjuntos de regras e um sistema para gerenciamento de algoritmos; 2º) identificar e propor estruturas de dados adequadas para manipulação de conjuntos, conceitos formais e reticulados conceituais; 3º) implementar novos algoritmos, em especial, algoritmos para geração de reticulados conceituais frequentes e algoritmos para extração de regras de associação; 4º) selecionar bases de dados reais e aplicar a análise formal, através do arcabouço, demonstrando assim as potencialidades da análise formal como ferramenta para análise quantitativa e qualitativa de bases de dados.

Referências

[1] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. *American Association for Artificial Intelligence*, pages 307–328, 1996.

[2] Gabriela Arévalo, Anne Berry, Marianne Huchard, Guillaume Perrot, and Alain Sigayret. Performances of galois sub-hierarchy-building algorithms. In *ICFCA'07: Proceedings of the 5th international conference on Formal concept analysis*, pages 166–180, Berlin, Heidelberg, 2007. Springer-Verlag.

[3] Karell Bertet, Stéphanie Guillas, and Jean-Marc Ogier. Extensions of Bordat's algorithm for attributes. In *Fifth International Conference on Concept Lattices and Their Applications*, pages 34–45, 2007.

[4] J. P. Bordat. Calcul pratique du treillis de galois d'une correspondance. *Mathématiques et sciences humaines*, 24(96):31–47, 1986.

[5] Frank Buchli. Detecting Software Patterns using Formal Concept Analysis. Diploma thesis, University of Bern, September 2003.

[6] C. Carpineto and G. Romano. *Concept Data Analysis: Theory and Applications*. John Wiley & Sons, England, 2004.

[7] Claudio Carpineto and Giovanni Romano. Galois: An order-theoretic approach to conceptual clustering. In *ICML*, pages 33–40, 1993.

[8] Michel Chein. Algorithme de recherche des sous-matrices premières d'une matrice. *Bull. Math. Soc. Sc. Math. de Roumanie*, 1(13):21–25, 1969.

[9] B. Davey and H. Priestley. *Introduction to lattices and order.* Cambridge University Press, Cambridge, England, 1990.

[10] Sérgio M. Dias. Algoritmos para geração de reticulados conceituais. Master's thesis, Federal University of Minas Gerais (UFMG), Institute of Exact Sciences, Department of Computer Science, Belo Horizonte, Minas Gerais, Brazil, 2010. in portuguese.

[11] Sérgio M. Dias and Newton J. Vieira. Reducing the size of concept lattices: The JBOS approach. In *Proceedings of the 7th International Conference on Concept Lattices and Their Applications*, volume 672, pages 80–91, Seville, Spain, 2010.

[12] Huaiguo Fu and Engelbert Mephu Nguifo. Partitioning large data to scale up lattice-based algorithm. In *International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 537–544, 2003.

[13] Huaiguo Fu and Engelbert Mephu Nguifo. A lattice algorithm for data mining. Technical report, Université d' Artois-IUT de Lens, 2004.

[14] Bernhard Ganter. Two basic algorithms in concept analysis. Technical report, TU Darmstadt, Germany, 1984.

[15] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, Germany, 1999.

[16] R. Godin, R. Missaoui, and H. Alaoui. Learning algorithms using galois lattice structure. In *Proceedings of the IEEE International Conference on Tools for Artificial Intelligence*, pages 22–29, 1991.

[17] Robert Godin, Rokia Missaoui, and Hassan Alaoui. Incremental concept formation algorithms based on galois (concepts) lattices. *Computation Intelligence*, 11, 1995.

[18] Robert Godin, Eugene Saunders, and Jan Gecsei. Lattice model of browsable data spaces. *Inf. Sci.*, 40(2):89–116, 1986.

[19] A. Guénoche. Construction du treillis de galois d'une relation binaire. *Mathématiques et Sciences Humaines*, (109):41–53, 1990.

[20] Nicolas Jay, François Kohler, and Amedeo Napoli. Analysis of social communities with iceberg and stability-based concept lattices. In *ICFCA*, pages 258–272, 2008.

[21] László Kovács. Concept lattice structure with attribute lattice. *Publication of University of Miskolc*, 10(8):985–1013, 2004.

[22] Sergei Kuznetsov. On computing the size of a lattice and related decision problems. *Order*, 18:313–321, 2001.

[23] Sergei Kuznetsov and Sergei Obiedkov. Comparing performance of algorithms for generating concept lattices. *Journal of Experimental & Theoretical Artificial Intelligence*, 14:189–216, April 2002.

[24] Y. Malgrange. Recherche des sous-matrices premères d'une matrice à coeffientes binaires; application à cetains problémes de graphe. In *Deuxième congrès de L'AFCALTI*, pages 231–242, 1962.

[25] Ben Martin and Peter W. Eklund. From concepts to concept lattice: A border algorithm for making covers explicit. In *ICFCA*, pages 78–89, 2008.

[26] Lhouari Nourine and Olivier Raynaud. A fast algorithm for building lattices. *MInformation Processing Letters*, 71:199 – 204, September 1999.

[27] Uta Priss. Formal concept analysis software, 2009. <http://www.upriss.org.uk/fca/fcasoftware.html> último acesso Janeiro de 2009.

[28] Sheng-Yong Qiao, Shuo-Pin Wen, Cai-Yun Chen, and Zhi-Guo Li. A fast algorithm for building concept lattice. In *Machine Learning and Cybernetics International Conference*, volume 1 Issue , 2-5, pages 163 – 167, 2003.

[29] Gerd Stumme. Formal concept analysis on its way from mathematics to computer science. In *ICCS*, Lecture Notes in Computer Science, pages 2–19, London, UK, 2002. Springer-Verlag.

[30] Petko Valtchev and Rokia Missaoui. A partition-based approach towards constructing galois (concept) lattices. *Discrete Mathematics*, 256:801–829, 2002.

[31] R. Wille. Restructuring lattice theory: an approach based on hierarchies of concepts. *I. Rival (Ed.): Ordered Sets*, pages 445–470, 1982.

[32] Serhiy A. Yevtushenko. System of data analysis "concept explorer". In *Proceedings of the 7th national conference on Artificial Intelligence KII-2000*, pages 127–134, Russia, 2000.

[33] L. E. Zárate, Sérgio M. Dias, and Mark A. J. Song. Fcann: A new approach for extraction and representation of knowledge from ann trained via formal concept analysis. *Neurocomputing*, 71:2670–2684, 2008.

[34] Luis E. Zárate and Sérgio M. Dias. Qualitative behavior rules for the cold rolling process extracted from trained ann via the fcann method. *Eng. Appl. AI*, 22(4-5):718–731, 2009.